Developing for Performance

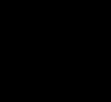
Session 906

Developing for Performance

Joe Sokol
Manager, CoreOS Performance Group

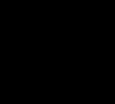
Introduction

- Mac OS X Performance Philosophy
  - “Improving system performance is like a diet, it’s a never-ending battle”
- To win this battle:
  - Use system resources efficiently
    - When quiescent
    - When active
  - Pay attention to the fundamentals
  - There are no magic APIs



What You Will Learn

- Performance and efficiency are different concepts
- Fundamental elements of system performance
    - CPU
    - Memory
    - File System I/O
- How to evaluate use of system resources



Performance vs. Efficiency

- They are not synonymous
    - Performance is a result of efficiency
    - Performance does not guarantee efficiency
- The processes for improving both are similar
- Why you should care

What Is Performance?

- Speed at which operations complete
- Perceived by the user
- Performance metrics
    - Throughput—bytes per second
    - OPS—operations per second
    - Latency—response time

What Is Efficiency?

- Cost to complete an operation
- Minimize use of system resources
- Efficiency metrics
    - CPU usage
    - Memory footprint
    - I/O frequency

System Performance

- Result of individual app and OS efficiencies
  - Heavily affected by working set size
- Affects user perception of app performance
- How do you improve it?

Fundamentals

- CPU
- Memory
- File System I/O

CPU

- Two models that are used
  - Good: Event driven
    - Only ‘real’ work is done
    - App can become totally ‘idle’
  - Bad: Polling
    - Usually no ‘real’ work to do
- Causes memory and possibly disk to be accessed

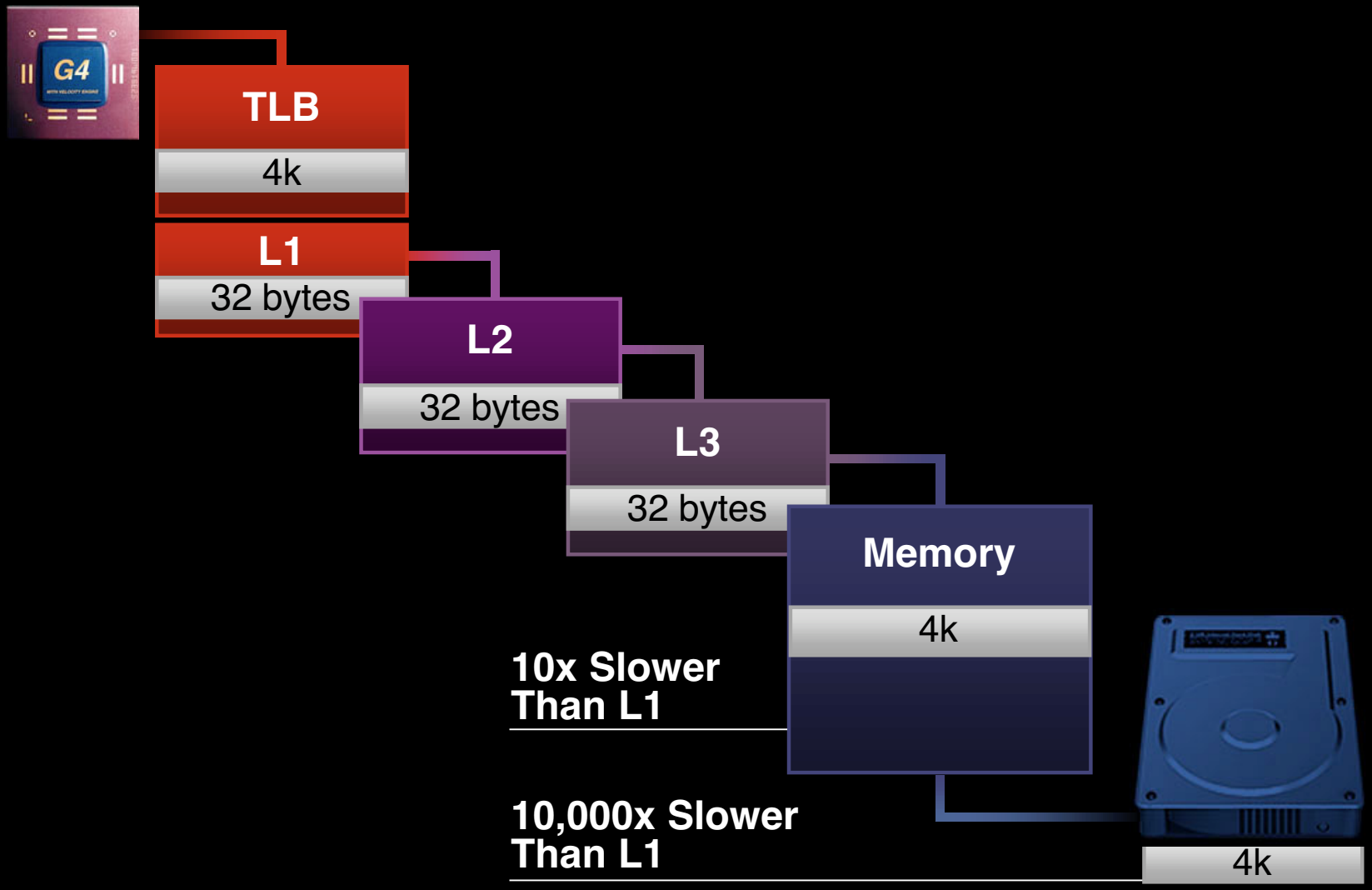
Why Event Driven?

- Working set minimized
  - Less paging = more responsiveness
- System can aggressively manage power
  - Battery lasts longer and
  - Fan active less often
- More CPU cycles available to do ‘real’ work

Why Not to Poll

- Larger working set is heated constantly
  - More contention for memory
  - More paging = sluggish system
- System may never become idle
  - Power management compromised
- Inefficient use of system resources

Touching Memory



Fundamentals

- CPU
- Memory
- File System I/O

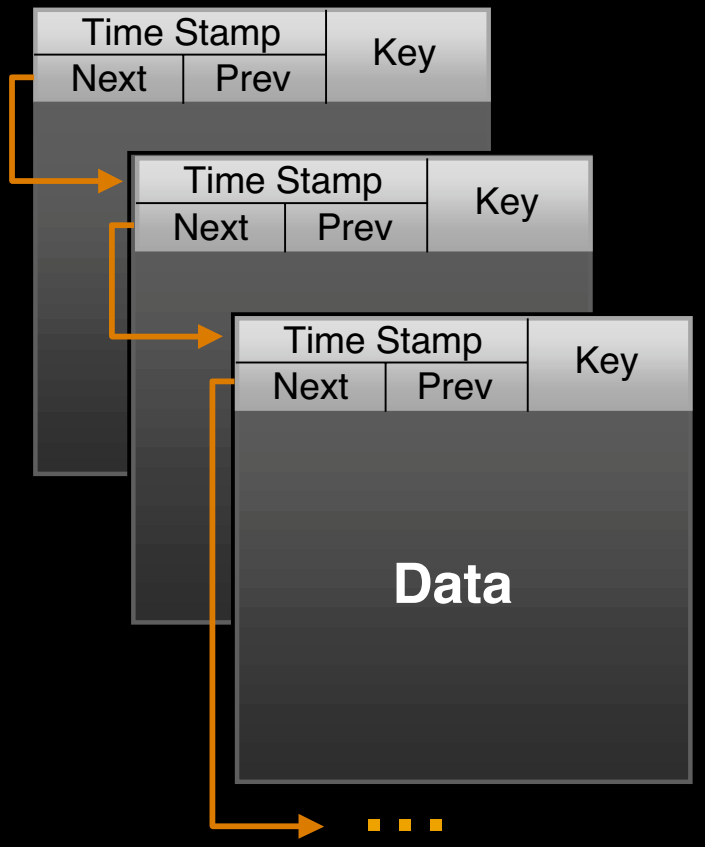
Memory

- Working set = pages used in RAM and on disk
  - heap and code
  - Every API used increases size
    - Window buffers can be large
  - Leaks increase size via heap fragmentation
- Long-term caches are always paged out
- Best performance when all pages are in RAM

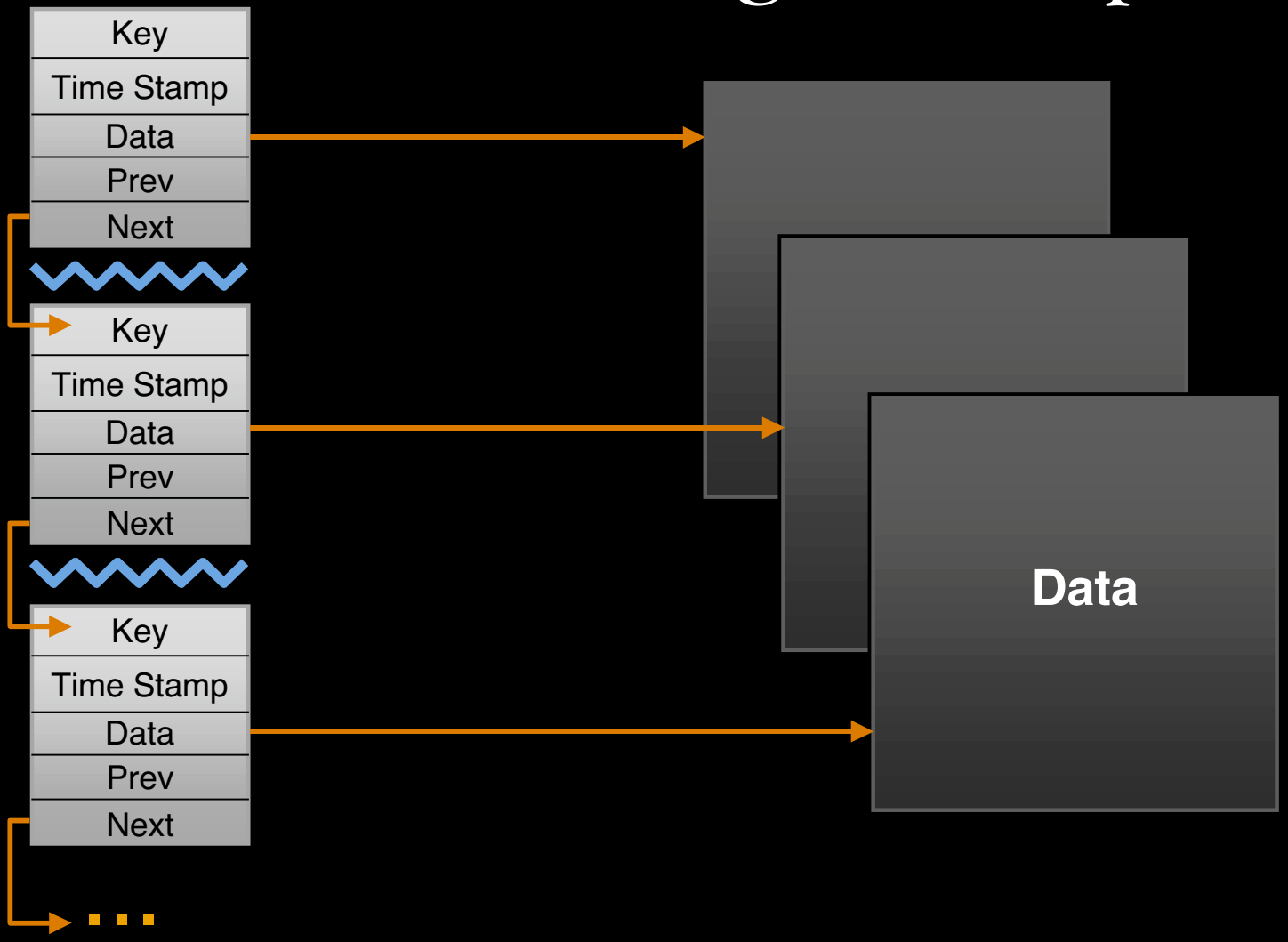
VM Interaction

- Typically 40% of pages in system are ‘dirty’
- ‘dirty’ pages will be ‘written’ to disk when
    - Unused pages are not available and
    - New memory is accessed or
    - File I/O is initiated or
    - Paged out memory is paged back in
- ‘clean’ pages can be ‘stolen’

Maximum Working Set Impact



Minimum Working Set Impact



Fundamentals

- CPU
- Memory
- File System I/O

Sequential vs. Random File I/O

- Sequential ~40Mbytes/sec on current disks
- Random ~0.4Mbytes/sec for 4K I/Os
- Larger I/Os are more efficient up to a point
  - I/O buffers consist of 'dirty' pages
  - Reasonable size is 128k
- File mapping for reading large files

File System MetaData

- Various MetaData will be written
  - When creating, deleting and writing
- Access times will be updated to disk
  - When enumerating directories
  - Reading files
- Polling models keep disk from sleeping

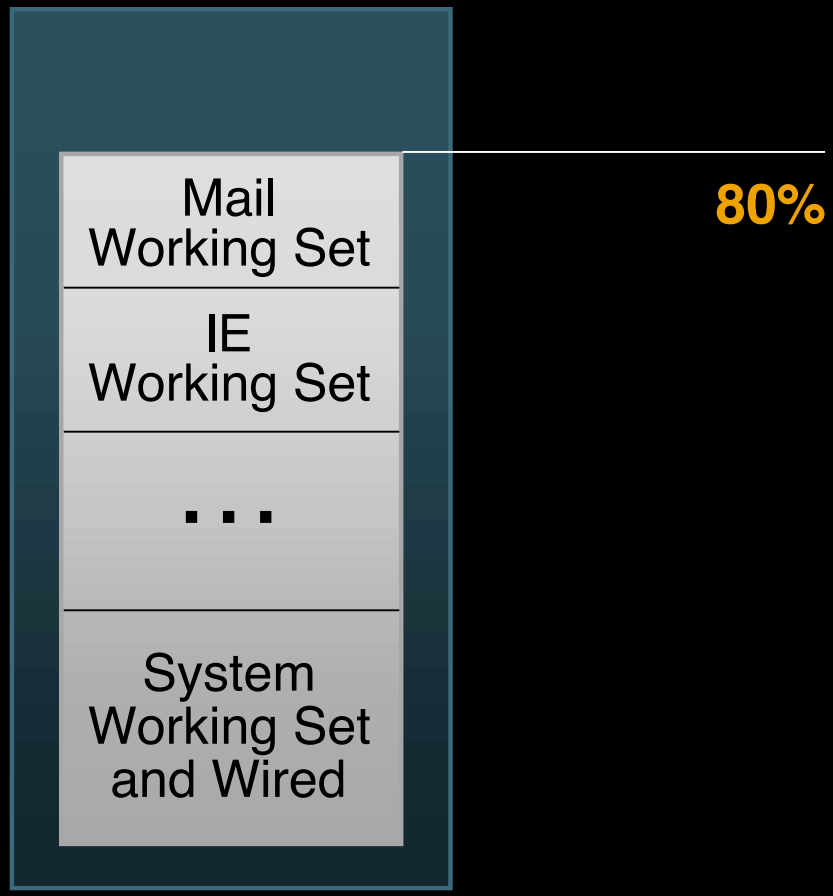
Network File Systems

- Files may reside on network storage
- Network performance unpredictable
- Polling models overwhelm networks

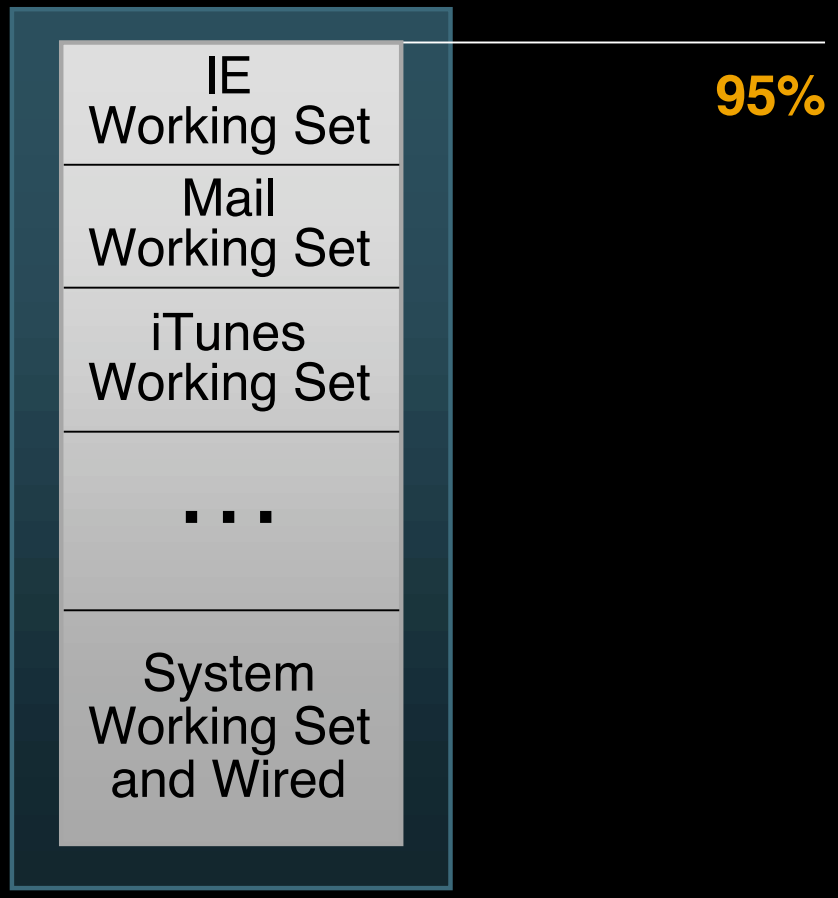
Fundamentals Are Important

- Your App does not run in isolation
  - Your App affects and is affected by other Apps
  - App efficiency -> system performance
- Minimizing working set size is critical

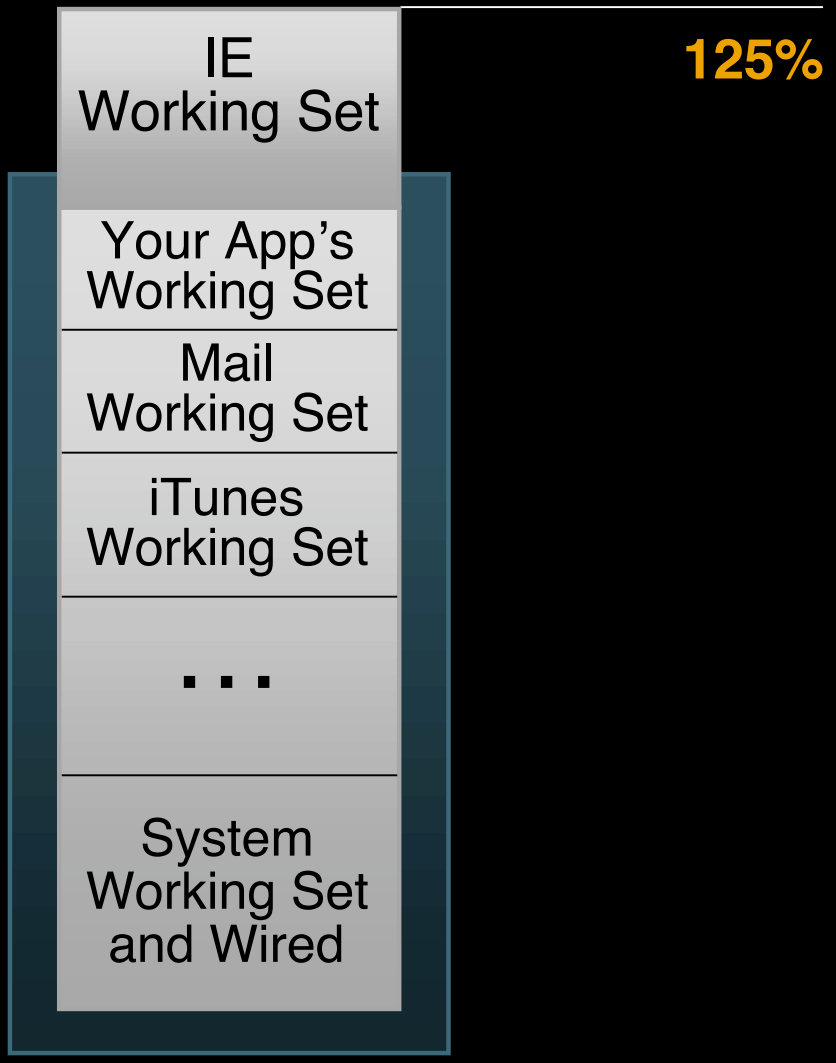
Working Set Impact



Working Set Impact



Working Set Impact



System Resource Diet

- Minimize your working set
  - Lazy initialization and event driven
- Don’t optimize at expense of efficiency
  - Your app may end up running slower
- Monitor efficiency during development
  - CPU usage
  - Memory footprint
  - File System I/O
- Don’t test in isolation

Improving Efficiency

- Set goal . . reduce memory footprint
- Identify metric . . heap size
- Measure it
- Analyze and apply changes
- Iterate until goal is met

Improving Performance

- Set goal . . improving throughput
- Select metric . . Mbytes/sec, Frames/sec
- Measure it
- Analyze and apply changes
- Iterate until goal is met

CPU Usage

- Monitoring
  - top -d or -a
- Analysis
  - Sampler or sample
  - Thread Viewer
  - Shikari
  - MONster

Memory Footprint

- Monitoring
  - top or “top -w”
  - leaks
- Analysis
  - heap
  - MallocDebug
  - ObjectAlloc

File System I/O

- Monitoring
  - fs_usage
  - top -d
- Analysis
  - Sampler
  - fs_usage

In Closing

- Applications that perform efficiently should be your goal
- Memory footprint is the ‘enemy’
- The hardest part is getting started
  - Lots of great tools available
- Customers will definitely notice

For More Information

- "Mac OS X: System Overview"
  "Mac OS X: Performance"
  on disk:
  /Developer/Documentation/Essentials/
  orderable in print from the web:
  http://developer.apple.com/techpubs/
- Apple Developer Connection tools page
  http://developer.apple.com/tools/

Roadmap

907 Compiler Developments at Apple: See the tools to optimize your application	Room J **Fri., 10:30am**
FF015 Development Tools: Make your thoughts known	Room J1 **Fri., 3:30pm**
909 Debugging in Mac OS X: Learn about gdb and debugging techniques	Hall 2 **Fri., 5:00pm**

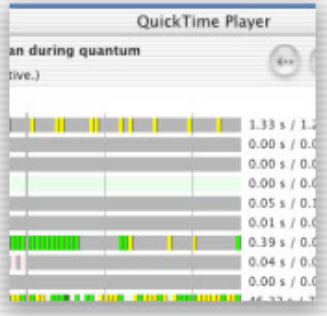
Who to Contact

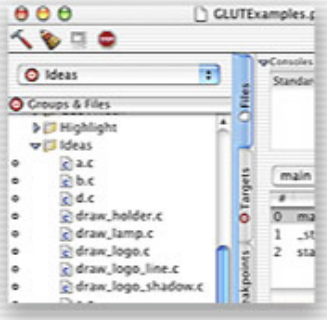
Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

Development Tools Engineering Feedback
macosx-tools-feedback@group.apple.com

http://developer.apple.com/wwdc2002/urls.html

Q&A





Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002